2019 年度 修士論文

ハイブリッドシステムシミュレータ HyLaGI の 電気回路例題への利用における問題の解決

提出日： 2020 年 1 月 29 日
指導： 上田 和紀 教授
研究指導名: 並列知識情報処理研究

早稲田大学大学院 基幹理工学研究科
情報理工・情報通信専攻

学籍番号：5118F051-1
渋井 隆弘

概要

HyLaGI はハイブリッドシステムモデリング言語 HydLa で書かれたプログラムを実行するためのハイブリッド制約記号シミュレータである．ハイブリッドシステムは時間の経過に伴って状態変数が連続変化と離散変化を起こすを持つシステムである．HyLaGI はバックエンドとして Mathematica を使用し制度保証計算を行い，変数の軌道を数式として求める．本研究ではハイブリッドシステムの中でも特に，電気回路系の例題を従来の HyLaGI を用いてシミュレートする際における問題点及びその解決方法を考察する．さらにその解決方法を従来の HyLaGI に実装として反映し，具体的にどのような電気回路をモデリング・解析できるかを調査した．

まず，従来の HyLaGI を用いて行う電気回路の解析を行う際に生じる問題を原因を特定し改善を試みた．回路素子を流れる電流や両端の電位差についての代数方程式や微分方程式を連立させたものは一般に DAE（微分代数方程式）と呼ばれる連立方程式で，ODE（線形微分方程式）よりも解くことが難しい．従来の HyLaGI における数式処理の実装は DAE を解くことを想定しておらず，電気回路が正しくモデリングされているにもかかわらず解析に失敗する場合がある．そこで，HyLaGI における数式処理を DAE に対応したものに改良することで，電気回路の解析に失敗する問題が解消された．

次に，数式処理システム Maple を用いて解析にかかる時間の短縮を試みた．回路が複雑になる（例えばループの数が増える）と解の数式が複雑になり，Mathematica による数式処理にかかる時間が非常に長くなってしまう．そこで，記号解を求めることに最適化された数式処理システム Maple を HyLaGI のバックエンドとして利用できるようにし，数式処理にかかる時間の短縮を試みた．

改善された HyLaGI を用いることでどのような目的で電気回路の解析ができるかを実験・考察した．電気回路系のハイブリッドシステムとしては電気回路中のスイッチの開閉やパルス信号の応答などが挙げられる．従来の HyLaGI を用いて解析すると，解析に失敗したり解析できても非常に長い時間がかかる電気回路の例題を短時間で解くことができるようになった．例えば交流入力信号の周波数を文字（パラメタ）として解析すると従来の HyLaGI では解析できず，Mathematica の処理を改善したものでは解析できるが時間がかかる．Maple を導入した HyLaGI では短時間で解析することができる．

目次

図目次

第 1 章

はじめに

1.1　研究の背景と目的

ハイブリッドシステム [1] とは，時間経過に伴い状態変数が連続変化と離散変化を起こすシステムである．ハイブリッドシステムの考え方の例として，硬い地面に落下し跳ね返る物体の軌道は物体が空中にいる間は運動方程式にのっとり連続的に変化するが，物体が地面に接触した瞬間に物体の速度の向きと大きさが離散的に変化するものとして扱うことが挙げられる．実際には非常に短い時間の中で連続的に速度が変化しているが，これを離散変化とみなすことで軌道の計算が簡単になり，自然現象をモデル化し解析する上で役に立つというのがハイブリッドシステムの考え方である [4]．この考え方は，力学系，電磁気学系を初めとする物理学系全般を始め，制御工学などの分野に応用が可能である．

ハイブリッドシステムをモデリングし，シミュレーションするための言語として HydLa[1] が存在する．HydLa プログラムの実行結果はプログラム中の変数の軌道を時刻 t で表される数式すなわち時刻 t の関数として求めたものである．HydLa を用いて解析できるハイブリッドシステムの例としてアナログ電気回路系のシステムが挙げられ，実行結果から回路素子を流れる電流や両端の電位差の時間変化を調べることが可能である．本論文中では，アナログ電気回路を単に電気回路と呼ぶことにする．

HydLa で記述されたプログラムを実行する処理系として HyLaGI[2] が開発されている．HyLaGI は C++ で開発されており，現在 3 万行程度の規模のソフトウェアである．HyLaGI は数式処理のためのバックエンドとして Mathematica を使用し，精度保証計算を行っている．従来の HyLaGI で電気回路をモデリングした HydLa プログラムを解く際，回路素子の両端の電位差と流れる電流の関係やキルヒホッフの電流則・電圧則から導かれる方程式を連立したものを解く過程で処理に失敗し，モデリングに誤りがないにもか

かわらず実行結果を得ることができない問題が存在する．

電気回路システムを解析する上で，微分代数方程式（DAE）と呼ばれる特殊な方程式について理解を深めなければならない．本論文中では，微分代数方程式を DAE と呼ぶことにする．線形な微分方程式と線形な代数方程式を連立したものは一般的に DAE であり常微分方程式（ODE）とは異なるため，扱いに注意が必要である．詳しい定義などは後の章で説明する．前述の問題が発生する原因は従来の HyLaGI に DAE を解くための処理が実装されていないことであり，DAE を解くための処理を行うように HyLaGI を改善することで解決する．

別の問題として，解析する回路の形がある程度複雑になると変数の軌道を示す数式が複雑になり，数式処理を担う Mathematica システムの負荷が大きくなる．すると，HyLaGI が結果を出すまでにかかる時間が非常に長くなったり，メモリが足りなくなり動作が完全に止まってしまうこともある．Mathematica はある程度の誤差を認める数値計算やデータサイエンス向けのライブラリなど多目的な機能を備えた汎用的なシステムとして開発されており，数式処理能力に特化したシステムではない．そこで，HyLaGI の実行過程における数式処理に記号計算に最適化されているシステムを用いて複雑な数式を高速に処理するという方法が考えられる．本研究では，記号計算に最適化されている代表的なシステムとして Maple を用いる．

ここまでに述べられた問題が解決されると，実用的な電気回路の解析や検証を行うことができるようになる．HydLa は離散変化を扱えるため，入力電源として，単なる直流，交流だけでなくステップ入力，パルス波，三角波，矩形波，のこぎり波などの形の電圧源に対する応答を調べることができる．入力電源の他に，スイッチの開閉やダイオードに対する応答など電気回路系において離散変化とみなせる現象を調べることができる．HydLa プログラムの実行結果は変数の軌道を時刻を表す変数 t の関数として求めたものであるから，極限 $t \to \infty$ を求めることで無限大時間後の回路の様子を調べることができる．

1.2　論文構成

2 章では HydLa で電気回路をモデリングする方法と HyLaGI による実行結果にについて解説する．3 章では DAE（微分代数方程式）について説明し，従来の HyLaGI の問題点と処理を改善する方法を説明する．4 章では数式処理システム Maple を HyLaGI のバックエンドとして利用する方法を説明する．5 章では例題を用いて従来から改善された HyLaGI の性能を評価・考察する．6 章では本論文を総括し，今後の課題を示す．

第 2 章

HydLa を用いた電気回路の解析方法と従来の HyLaGI で生じる問題点

この章では，HydLa 言語およびその処理系 HyLaGI についての解説および，HydLa 用いて電気回路を解析する方法を説明する．

2.1 HydLa 言語

HydLa は，ハイブリッドシステムをモデリングするための宣言型言語である．状態変数の満たすべき制約を常微分方程式や不等式として記述する．各変数は全て時刻 t の関数として求まる．また必要に応じて制約の優先順位を制約階層として宣言し，制約階層内で全ての制約を満足する解が存在しない場合は場合はより上にある制約のみを満足する解を求める．

2.1.1 ハイブリッドシステム

ハイブリッドシステムは，時間経過に伴い状態変数が連続変化と離散変化を起こすシステムである．電気回路系において連続変化とみなせる現象として，抵抗・インダクタ・キャパシタの両端の電位差と電流の関係や正弦波信号があげられる．一方，離散変化とみなせる代表的な現象としてスイッチの開閉や，ダイオード，パルス信号などがある．よって電気回路系はハイブリッドシステムであるといえる．

2.1.2　HydLa プログラムの例

初めに例として，図 2.1 に電気回路で使うパルス信号をモデリングするための HydLa プログラムを示す．

```
1  INIT <=> vin=0.
2  VIN <=> [](vin'=0).
3  UP <=> [](t=1 => vin=1).
4  DOWN <=> [](t=2 => vin=0).
5
6  INIT, (VIN << (UP, DOWN)).
```

図 2.1　パルス信号

1 行目から 4 行目は各制約を INIT，VIN，UP，DOWN と定義してモジュール化している．この INIT，VIN，UP，DOWN を制約モジュールと言う．

INIT は $t = 0$ における vin の初期値は 0 であるという意味である．HydLa の初期時刻は必ず $t = 0$ である．

VIN は時刻に関係なく vin' の値は 0 であるという意味である．vin' は vin の時間微分値を表す．また [] は時相論理の「常に（always)」という意味の量化子である．

UP および DOWN はガード条件付きの制約で，$t = 1$ および $t = 2$ の瞬間のみ vin の値が制約される．HydLa では t は予約変数で時刻を表す．ガード条件が成立する時刻は一般に 0 ではないから [] をつける必要がある．

6 行目は各制約モジュールの制約階層を定義している．制約 A に優先する制約が存在しないとき A は required であるといい，求められる解は required な制約を必ず満足していなければならない．図 2.1 の例では，INIT，UP，DOWN が required な制約モジュールで，UP，DOWN は VIN に優先する制約モジュールである．

次に，図 2.1 のプログラムを HydLa 専用の処理系 HyLaGI で実行した際，どのようにして解が求められるのかを説明する．$t = 0$ においては全ての制約モジュールを満足できるため解は vin=0，vin'=0 となる．UP，DOWN についてはガード条件が満たされないときは右辺の制約を満たさなくても全体としては真であることに注意する．$0 < t < 1$ においても全ての制約モジュールを満足できるため解は vin'=0 となる．INIT は $t = 0$ における制約であるため $t = 0$ 以外では vin の値に関係なく真である．ここで $t = 0$ と

$0 < t < 1$ を組み合わせて $0 \leq t < 1$ における次の微分方程式を解き解は $vin(t) = 0$ となる．

$$\begin{cases} vin(0) & = 0 \\ \dfrac{d}{dt}vin(t) & = 0 \ (0 \leq t < 1) \end{cases}$$

$t = 1$ においては UP のガード条件が真となるために，全ての制約モジュールを満足しようとすると，VIN（vin は時間変化しない）と UP（vin の $t = 1$ における左極限値に関係なく vin=1 である）が矛盾する．その為 VIN を満足すべき条件から除外し，INIT，UP，DOWN を満足する解は vin=1 である．$1 < t < 2$ においては全ての制約モジュールを満足できるため解は vin'=0 となる．ここで $t = 1$ と $1 < t < 2$ を組み合わせて $1 \leq t < 2$ における次の微分方程式を解き解は $vin(t) = 1$ となる．

$$\begin{cases} vin(1) & = 1 \\ \dfrac{d}{dt}vin(t) & = 0 \ (1 \leq t < 2) \end{cases}$$

$t = 2$ においては DOWN のガード条件が真となるために，全ての制約モジュールを満足しようとすると，VIN と DOWN が矛盾する．その為 VIN を満足すべき条件から除外し，INIT，UP，DOWN を満足する解は vin=0 である．$2 < t$ においては全ての制約モジュールを満足できるため解は vin'=0 となる．ここで $t = 2$ と $2 < t$ を組み合わせて $2 \leq t$ における次の微分方程式を解き解は $vin(t) = 0$ となる．

$$\begin{cases} vin(2) & = 0 \\ \dfrac{d}{dt}vin(t) & = 0 \ (2 \leq t) \end{cases}$$

以上の結果をすべてまとめると結果は次のようになる．

$$vin(t) = \begin{cases} 1 & (1 \leq t < 2) \\ 0 & (0 \leq t < 1, 2 \leq t) \end{cases}$$

次に図 2.1 を利用してパルス信号に対する電気回路の応答を調べるプログラムの例を紹介する．ここでは，説明のために簡単な回路としてパルス信号を発生させる電圧源，抵抗，キャパシタ（コンデンサー）の 3 つの回路素子からなる単一ループの例題を 2.2 示す．

1 行目は時刻に等しい変数 timer を定義するモジュール定義で 12 行目に宣言されており，図 2.1 のように直接 t を用いずに時刻を扱うことができる．2 行目から 4 行目はアナログ回路素子抵抗（R），インダクタ（L），キャパシタ（C）の性質を宣言するための引数あり制約のマクロ定義であり，13 行目に回路中に存在する抵抗とキャパシタの性質を宣言している．R(v1,i,r) の記述は，抵抗値 r の抵抗が存在してその両端の電位差は

```
1  TIMER <=> timer=0 & [](timer'=1).
2  R(v,i,r) <=> [](v=r*i).
3  L(v,i,l) <=> [](v=l*i').
4  C(v,i,c) <=> [](c*v'=i).
5  PAR(a,b,c) <=> a<b<c & [](b'=0).
6  INIT <=> vin=0 & v2=0.
7  VIN <=> [](vin'=0).
8  UP <=> [](timer=1 => vin=1).
9  DOWN <=> [](timer=2 => vin=0).
10 CIRCUIT <=> [](vin=v1+v2).
11
12 TIMER,
13 R(v1,i,r), C(v2,i,c),
14 PAR(0.5,r,1.5), PAR(0.5,c,1.5),
15 INIT,
16 VIN << (UP, DOWN),
17 CIRCUIT.
```

図 2.2　電気回路にパルス信号を入力する

v1，流れる電流が i であることを宣言し，C(v2,i,c) の記述は，容量 c のキャパシタが存在してその両端の電位差は v2，流れる電流が i であることを宣言している．5 行目は定数パラメタを宣言するためのマクロ定義である．例えば，PAR(0.5,r,1.5) の記述は 0.5<r<1.5 & [](r'=0) の記述と等価であり，時刻 $t = 0$ における r の値は 0.5 より大きく 1.5 未満かつ r の微分値が常に 0 である即ち定数であることを意味し，r が 0.5 より大きく 1.5 未満の定数であることを宣言している．ここで [](0.5<r<1.5) と記述するだけでは，r の値は常に 0.5 より大きく 1.5 未満の範囲に存在するが時間の変化に伴う変化が制限されないので定数とならないので注意が必要である．

2.2　処理系 HyLaGI

HyLaGI は HydLa 専用の処理系である．HydLa プログラムの解を時刻 t の関数として求める．HyLaGI は精度保証された解軌道のためにバックエンドとして技術計算システ

ム Mathematica [9] を使用している．HyLaGI が出力する結果は LTspice[6][8] などの回路シミュレータとは異なり，各変数を時刻 t についての数式として記号的に求めることができるため誤差が生じない．

2.2.1　HydLa プログラムの実行結果

HyLaGI で図 2.2 の HydLa プログラムを実行した際の結果の一部を図 2.3 に示す．

```
1  t : 2->Infinity
2  c : p[c, 0, 1]
3  i : E^((1+t*(-1))*p[c, 0, 1]^(-1)*p[r, 0, 1]^(-1))*(-1)*(E^(p[c, 0,
       1]^(-1)*p[r, 0, 1]^(-1))+(-1))*p[r, 0, 1]^(-1)
4  r : p[r, 0, 1]
5  timer : t
6  v1 : E^((2+t*(-1))*p[c, 0, 1]^(-1)*p[r, 0, 1]^(-1))*(E^(-1*p[c, 0,
       1]^(-1)*p[r, 0, 1]^(-1))+(-1))
7  v2 : E^((2+t*(-1))*p[c, 0, 1]^(-1)*p[r, 0, 1]^(-1))*(1+E^(-1*p[c, 0,
       1]^(-1)*p[r, 0, 1]^(-1))*(-1))
8  vin : 0
9  c' : 0
10 r' : 0
11 timer' : 1
12 v2' : E^((1+t*(-1))*p[c, 0, 1]^(-1)*p[r, 0, 1]^(-1))*(-1)*(E^(p[c, 0,
       1]^(-1)*p[r, 0, 1]^(-1))+(-1))*p[c, 0, 1]^(-1)*p[r, 0, 1]^(-1)
13 vin' : 0
```

図 2.3　実行結果

1 行目は解軌道全体のうち，どの時刻区間についての解であるかを示している．2 行目以降は宣言されている変数や s の微分値が区間内で時刻 t の関数としてどのように表されるかを表している．`p[r, 0, 1]` や `p[c, 0, 1]` はパラメタであり，`p[a, m, n]` は変数 `a` の m 回微分の第 $n$phase の値である．例えば，抵抗の両端に加わる電圧（`v1`）を数式で表すと

$$e^{-\frac{t-1}{rc}} - e^{-\frac{t-2}{rc}}$$

となる．

第 3 章

DAE（微分代数方程式）の性質と従来の HyLaGI における数式処理の問題点

本章では，DAE（微分代数方程式）の定義および性質を初めに説明する．次に，従来の HyLaGI における数式処理の問題点および解決方法を 2 つ提案する．また，DAE に関連する研究を紹介する．

3.1 DAE（微分代数方程式）

DAE（Differential Algebraic Equations，微分代数方程式）は簡単に言うと微分方程式と代数方程式を連立させたものである．

3.1.1 DAE の定義

次の形式で表される連立微分方程式を考える．

$$A\boldsymbol{x}'(t) + B\boldsymbol{x}(t) = F \tag{3.1}$$

ただし，A，B は各要素が定数または独立変数 t についての関数である行列，F は各要素が定数または独立変数 t についての関数であるベクトル，$\boldsymbol{x}(t)$ は t についての未知関数ベクトルである．このとき t の値に関係なく $\det(A) = 0$ ならば DAE である．（$\det(A) \neq 0$ ならば常微分方程式である．）

例えば，微分方程式と代数方程式を連立させた方程式

$$\begin{cases} x'(t) = y(t) \\ x(t) + y(t) = 1 \end{cases} \tag{3.2}$$

は式 3.1 において $\boldsymbol{x}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$, $A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 0 & -1 \\ 1 & 1 \end{pmatrix}$ $F = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ の場合とした場合であり $\det(A) = 0$ であるから式 3.2 は DAE である．また，微分方程式のみの連立方程式も DAE であることがある．例えば次の連立方程式

$$\begin{cases} x'(t) + y'(t) = x(t) \\ x'(t) + y'(t) = y(t) \end{cases} \tag{3.3}$$

は式 3.1 において $\boldsymbol{x}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$, $A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$, $B = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$ $F = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ の場合とした場合であり $\det(A) = 0$ であるから式 3.3 は微分方程式のみの連立方程式でありながら DAE である．

3.1.2　DAE の性質

常微分方程式の一般解に含まれる一般解の自由度（任意定数の数）は変数の数に等しいという特徴がある．一方，DAE は変数の数が等しい常微分方程式に比べ一般解の自由度が小さい，つまり一般解に含まれる任意定数の数が少ない．次の微分方程式

$$\begin{cases} x'(t) - y(t) = 0 \\ y'(t) + x(t) = 0 \end{cases} \tag{3.4}$$

は式 3.1 において $\boldsymbol{x}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$, $A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $B = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ $F = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ の場合とした場合であり $\det(A) \neq 0$ であるから式 3.4 は常微分方程式である．(3.4) の一般解は

$$\begin{cases} x(t) = A\cos t + B\sin t \\ y(t) = B\cos t - A\sin t \end{cases} \quad (A, B \text{ は任意定数}) \tag{3.5}$$

であり任意定数はの数は 2 つで変数の数と等しい．一方 (3.2) の一般解は

$$\begin{cases} x(t) = Ae^{-t} + 1 \\ y(t) = -Ae^{-t} \end{cases} \quad (A \text{ は任意定数}) \tag{3.6}$$

(3.3) の一般解は

$$\begin{cases} x(t) = Ae^{\frac{t}{2}} \\ y(t) = Ae^{\frac{t}{2}} \end{cases} \quad (A \text{ は任意定数}) \tag{3.7}$$

であり，任意定数の数が 1 つであり変数の数よりも少ない．この事実は DAE の一般解において異なる変数間に線形従属が存在していることを意味する．(3.6) は $x(t)+y(t)=1$，(3.7) は $x(t)=y(t)$ という関係があり，線形従属である．

3.2　HyLaGI の処理過程における DAE の存在

DAE は変数の数が等しい常微分方程式と比較して一般解の自由度が小さいという性質は初期値問題を解く際に注意を要する．そのため，HyLaGI を含め数式を扱うシステムを設計する者は DAE について理解する必要がある．

3.2.1　HyLaGI の数式処理における役割

HyLaGI のバックエンドとして用いられる Mathematica は汎用的なシステムとして開発されているため特定の使用方法について弱点がある．例えば，次の連立微分方程式を Mathematica の組み込み関数 DSolve に処理させても解くことができない．

$$\begin{cases} y(t)=x(t)^2 \\ \dfrac{dx}{dt}(t)=x(t) \\ x(0)=1 \end{cases} \tag{3.8}$$

実際，Mathematica カーネルを起動し，(3.8) を解こうとすると図 3.1 のように与えた式がそのまま帰ってくる．Mathematica は関数適用できなかった式は与えられた形のまま返すように設計されている．(`x'[t]` は `Derivative[1][x][t]` の略記法である．)

```
1 In[1]:= InputForm[DSolve[{y[t] == x[t]^2, x'[t] == x[t], x[0]
      == 1}, {x[t], y[t]}, t]]
2
3 Out[1]//InputForm= DSolve[{y[t] == x[t]^2, Derivative[1][x][t]
      == x[t], x[0] == 1}, {x[t], y[t]}, t]
```

図 3.1　DSolve で解くことのできない微分方程式

人間ならば，下 2 つの式のみを解いてから上の式に代入すればよいということがすぐに分かる．HyLaGI は連立微分方程式を解くために，Wolfram 言語（Mathematica システムを利用するための言語）で定義された関数 exDSolve を備えている．exDSolve のアル

ゴリズムを簡単に説明すると，まず初期条件式を除外し含まれる t についての未知関数が少ない順に方程式をソートしてリスト化し，次にリストの先頭から方程式を 1 つずつ集めてゆき方程式の数と t についての未知関数の数が等しくなったら，DSolve で解くというものである．(3.8) を解く場合，まず $\frac{dx}{dt}(t) = x(t)$ を解き DSolve で解き，その結果を用いて残りの部分を再び DSolve で解くことで処理が完了する．実際に図 3.2 の HydLa プログラムを HyLaGI で実行すると図 3.3 のように解くことができることがわかる．

```
1 [](y = x^2), [](x' = x), x = 0.
```

図 3.2　(3.8) を解くための HydLa プログラム

```
1  ------ Result of Simulation ------
2  ---------Case 1---------
3  ---------1---------
4  ---------PP 1---------
5  unadopted modules: {}
6  positive  :
7  negative  :
8  t : 0
9  x : 1
10 y : 1
11 x' : 1
12 ---------IP 2---------
13 unadopted modules: {}
14 positive  :
15 negative  :
16 t : 0->Infinity
17 x : E^t
18 y : E^(2*t)
19 x' : E^t
```

図 3.3　図 3.2 を HyLaGI で実行した結果

HydLa の文法は図 3.2 のように，モジュール化していない制約を制約階層に直接記述することを許可している．

3.2.2　HyLaGI が DAE を処理する際の問題

常微分方程式や DAE の初期値問題を解く際に解の自由度が異なると解が 1 つに決まるために必要な初期条件の数が異なるため，数式処理の過程で解こうとする微分方程式が DAE および常微分方程式のどちらなのかがわからないと問題が起こる．例えば，(3.6) における $x(t) + y(t) = 1$ という関係は与えられた方程式 3.2 中に明示されているが，(3.6) における $x(t) = y(t)$ という関係は方程式 3.3 中に明示されていない．実際にどのような問題が起こるかを調べるために図 3.4 に示す HydLa プログラムを従来の HyLaGI で実行すると，図 3.5 に示すエラーメッセージが出力されて解を求めることができずに実行が終了する．

```
1 []( x' + y' = x),
2 []( x' + y' = y),
3 0 < x < 2,
4 1 < y < 3.
```

図 3.4　従来の HyLaGI で実行すると問題が発生する HydLa プログラム

```
1 DSolve::bvnul: For some branches of the general solution, the
      given boundary conditions lead to an empty solution.
```

図 3.5　従来の HyLaGI で実行すると問題が発生する HydLa プログラム

図 3.5 は Mathematica の DSolve 関数に与えられた制約を満足できる解が存在しないときに出力される例外メッセージである．この問題の原因を調べるためには，HyLaGI の実行過程で DSolve が実行されるときの引数を調べる必要がある．

HyLaGI にはデバッグ出力機能があり，Mathematica の実行中の特定の地点における変数の値を調べることができる．調べた結果，原因となる DSolve の実行文と実行結果を図 3.6 に示す．

図 3.6 では次の DAE の初期値問題を解こうとするものである．ただし，`prev[px, 0]` は a，`prev[py, 0]` は b に置き換えている．

```
1 In[1]:= DSolve[{ux[t] == Derivative[1][ux][t] + Derivative[1][
      uy][t], uy[t] == Derivative[1][ux][t] + Derivative[1][uy][t
      ], ux[0] == prev[px, 0], uy[0] == prev[py, 0]}, {ux[t], uy[
      t]}, t]
2
3 DSolve::bvnul: For some branches of the general solution, the
      given boundary conditions lead to an empty solution.
4
5 Out[1]= {}
```

図 3.6　原因となる DSolve の実行文

$$\begin{cases} ux'(t) + uy'(t) = ux(t) \\ ux'(t) + uy'(t) = uy(t) \\ ux(0) = a \\ uy(0) = b \end{cases} \tag{3.9}$$

HyLaGI は Mathematica が問題を解きやすくするために，不等式を含む制約集合を解くために初めに不等式を除いた制約集合を，変数の初期値を `prev[px, 0]` 等の表記で与えて解き，その解と残りの不等式を連立させて解くことで与えられた制約集合全体についての解を得ている．従来の HyLaGI では関数 exDSolve の中で用いられる関数 solveByDSolve によって，各変数について式中に出現する最多微分階数を n とするとき 0 から $n-1$ 階微分の初期値を関数 DSolve に与えている．例えば，(3.9) では $ux(t)$，$uy(t)$ について式中に出現する最多微分階数が 1 であるから各変数の 0 階微分の初期値 $ux(0)$，$uy(0)$ を与えられている．(3.9) の解は (3.3) の解（図 3.7）を参考にすると $a = b$ の場合に限り $ux(t) = uy(t) = ae^{\frac{t}{2}}$ という解が存在することがわかる．しかし Mathematica は図 3.6 の 1 行目の入力が与えられた時，あくまで `ux[t]` と `uy[t]` について解く命令なので `prev[px, 0]` と `prev[py, 0]` の間に関係が生じるような解を認めずに解無しとするのである．

3.3　HyLaGI で DAE を解く方法

HyLaGI で不等式を除いた制約集合を解く段階において DAE を解くために 2 つの解決方法を提案する．1 つ目は関数 DSolve に与える初期条件を減らす方法，2 つ目は一般解を経由する方法である．各方法についての比較実験は章 5 で行う．なお，本節についての詳細は情報処理学会第 81 回全国大会にて発表を行った．[1]

3.3.1　関数 DSolve に与える初期条件を減らす方法（手法 1）

最初に，従来の HyLaGI が与える初期条件の集合の冪集合を求め，その各要素を要素数が大きい順にソートしたリストを作る．そのリストの先頭要素から順に DSolve に与える初期条件として用いて DAE が解けるか試行してゆき，解くことができた場合その解を採用して不等式の処理に移る．DSolve に与えた初期条件の集合が解が得られないものだった場合，解く DAE が複雑なものであっても短時間で例外を出力し次の要素を試すことができる．この処理を行うために図 3.7 に示すように関数 solveByDSolve を修正した．この方法を手法 1 とする．

3.3.2　一般解を経由する方法（手法 2）

もう一つの方法として一般解を求めてから任意定数を消去することで初期値問題の解を得る方法を考える．最初に，初期条件を与えずに関数 DSolve を実行し一般解を求め，得られた一般解に含まれる任意定数の数（＝解の自由度）を調べる．次に，解の自由度に等しい数の要素を持つ初期条件の部分集合のリストを作る．Mathematica の組み込み関数 Solve を用いて，リストの各要素について先頭要素からと一般解と連立させた代数方程式から任意定数を消去できるか調べ，消去できたらそれを初期値問題の解として採用する．この処理を行うために新たに図 3.8 に示すように関数 solveByDSolveAndSolve を作成し，関数 exDSolve 内で関数 solveByDSolve の代わりに用いる．この方法を手法 2 とする．

3.4　DAE に関連する研究

線形時不変回路を解析するための DAE（混合方程式と呼ばれる）の指数を下げることで数値計算時の誤差を小さくすることができるという内容の研究が存在する [3]．本研究

```
solveByDSolve[expr_, vars_] :=
solveByDSolve[expr, vars] = (* for memoization *)
Module[
  {ini = {}, inis, sol, derivatives, i},
  tVars = Map[(#[t])&, vars];
  derivatives = getTimeVariablesWithDerivatives[expr];
  For[i = 1, i <= Length[derivatives], i++,
    ini = Union[ini, createPrevRules[derivatives[[i]] ] ]
  ];
  tmp = expr;
  For[i = Length[ini], i >= 0, i--,
    inis = Subsets[ini, {i}];
    For[j = 1, j <= Length[inis], j++,
      sol = Quiet[
        Check[
          DSolve[Union[expr, inis[[j]]], tVars, t],
              overConstrained,
          {DSolve::overdet, DSolve::bvimp}
        ],
        {DSolve::overdet, DSolve::bvimp, DSolve::bvnul, Solve::svars,
            PolynomialGCD::lrgexp}
      ];
      If[Length[sol] > 0,
        Break[]
      ]
    ];
    If[Length[sol] > 0,
      Break[]
    ]
  ];
  For[i = 1, i <= Length[sol], i++,
    If[Count[Map[(timeConstrainedSimplify[Element[#[[2]], Reals]])&, sol
        [[i]] ], False] > 0,
      sol = Drop[sol, {i}];
      --i;
    ]
  ];
  If[Length[sol] > 0, sol, overConstrained]
];
```

図 3.7　solveByDSolve

も，問題の解決のために DAE の性質を理解する必要があり関連している．一方，本研究は HyLaGI による記号計算（数式を数式のまま解く）における問題点の解決を扱うため誤差の概念が無く，数式的な解を求めることができるかどうかや実行時間に焦点を当てているため，数値計算に関する研究と差別化される．

```
solveByDSolveAndSolve[expr_, vars_] :=
solveByDSolveAndSolve[expr, vars] = (* for memoization *)
Module[
  {ini, inis, sol = {}, solwithconstant, solofconstant, derivatives,
      prevs = {}, constants, i, j},
  tVars = Map[(#[t])&, vars];
  derivatives = getTimeVariablesWithDerivatives[expr];
  For[i = 1, i <= Length[derivatives], i++,
    prevs = Union[prevs, createPrevOnly[derivatives[[i]] ] ]
  ];
  tmp = expr;
  solwithconstant = Quiet[
    Check[
      DSolve[expr, vars, t],
          overConstrained,
      {DSolve::overdet, DSolve::bvimp}
    ],
    {DSolve::overdet, DSolve::bvimp, DSolve::bvnul, Solve::svars,
        PolynomialGCD::lrgexp}
  ];
  For[i = 1, i <= Length[solwithconstant], i++,
    ini = {};
    constants = Union[Cases[solwithconstant[[i]], C[_], {0, Infinity}]];
    For[j = 1, j <= Length[prevs], j++,
      ini = Append[ini, Simplify[prevs[[j]][0] /. solwithconstant[[i]] ]
          == makePrevVar[prevs[[j]] ] ]
    ];
    inis = Subsets[ini,{Length[constants]}];
    For[j = 1, j <= Length[inis], j++,
      solofconstant = Quiet[
        Check[
          Solve[inis[[j]], constants, Reals],
              overConstrained,
          {DSolve::overdet, DSolve::bvimp}
        ],
        {DSolve::overdet, DSolve::bvimp, DSolve::bvnul, Solve::svars,
            PolynomialGCD::lrgexp}
      ];
      If[Length[solofconstant] > 0,
        sol = Union[sol, Map[#[t]&, (solwithconstant[[i]] /.
            solofconstant ), {3}]];
        Break[]
      ]
    ];
  ];
  If[Length[sol] == 0, Return[overConstrained]];
  For[i = 1, i <= Length[sol], i++,
    If[Count[Map[(timeConstrainedSimplify[Element[#[[2]], Reals]])&, sol
        [[i]] ], False] > 0,
      sol = Drop[sol, {i}];
      --i;
    ]
  ];
  If[Length[sol] > 0, sol, overConstrained]
];
```

図 3.8　solveByDSolveAndSolve

第 4 章

Maple を用いた HyLaGI における数式処理

この章では初めに，数式処理システム Maple[10] について説明する．次に，Mathematicat と Maple の DAE 解析能力を比較する．また，HyLaGI の数式処理に Maple を利用するための方法，実装内容について説明する．

4.1 数式処理システム Maple

Maple は Maplesoft が開発する数式処理システムである．Maple に関する研究としてプロジェクト「ロボットは東大に入れるか」において数学の問題を解く目的で，富士通研究所が開発した Maple 用モジュール SyNRAC が使用されたことがある．

4.1.1 Mathematica と比較した Maple の特徴

Maple は記号計算（数式のまま計算）に最適化されており，この点においては Mathematica に対して優れていると言える．しかし，システムの規模が小さく大量の変数や制約を記憶することは苦手である．一方，Mathematica は汎用的なシステムとして開発されており，システムの規模が大きい．そのため，大量の変数や制約を記憶する用途に適し，実際に HyLaGI の実行過程において変数や制約集合の解を実行終了時に解を出力するまで記憶する役割を持つ．Mathematica および Maple の長所を活かすためには，HyLaGI において方程式を解く場面でのみ Maple を用いるのが良いと考えられる．

4.2　Mathematicat と Maple の DAE 解析能力を比較

Maple を HyLaGI に導入することで電気回路解析の助けになるかを図 4.1 に示す例題を用いて検証する

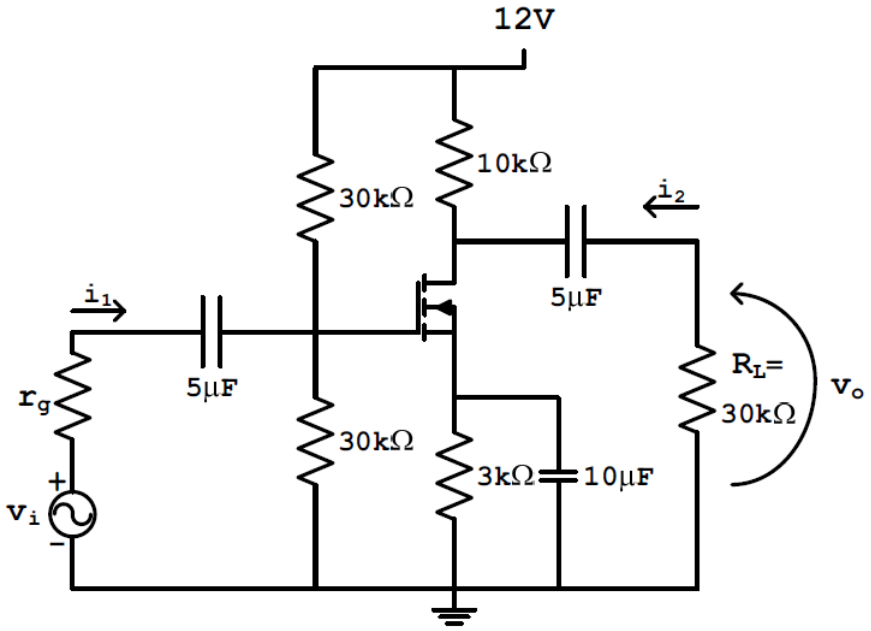

図 4.1　MOSFET を使用した信号増幅回路

図 4.1 は早稲田大学 2015 年度の講義「電子回路」のレポート課題より MOSFET を使用した信号増幅回路である．$r_g = 10[\mathrm{k}\Omega]$ とする．またゲート電圧と MOSFET を流れる電流は完全に比例するとしその値は 10[mS]=10[mAV^{-1}] とする．解析方法については卒業論文を参考にされたい．図 4.1 の回路において $v_i = \sin at$ としたときに解くことになる DAE の一般解について，Mathematica および Maple で解くのにかかる時間と解の char 数（空白や改行を除く）を調べる．

4.2.1　実行環境

CPU:AMD Ryzen Threadripper 1950X 3.4GHz
メモリ:64GB
OS:Ubuntu 18.04 LTS
Mathematica のバージョン:11.3

Maple のバージョン:Maple2015

4.2.2　Mathematica の DAE 解析能力

Mathematica における実行文を図 4.2 に示す.

```
1 Timing[
2 DSolve[{30*ui3[t] == uv3[t], 3*ui4[t] == uv4[t], ui5[t] == 10*
      Derivative[1][uv5][t], ui4[t] + ui5[t] == uids[t], 30*ui6[t
      ] == uv6[t], ui1[t] + ui6[t] == ui3[t], 10*ui7[t] == uv7[t
      ], ui2[t] + ui7[t] == uids[t], uids[t] == 10*uvgs[t], uv11[
      t] == -10*ui1[t], uv3[t] == uv4[t] + uvgs[t], uv4[t] == uv5
      [t], uv3[t] + uv6[t] == 12, Sin[t*prev[pa, 0]] + uv11[t] +
      uv12[t] == uv3[t], uvo[t] == -30*ui2[t], uv2[t] + uvo[t] ==
       uv4[t] + uvds[t], uv2[t] + uv7[t] + uvo[t] == 12, ui1[t] +
       5*Derivative[1][uv12][t] == 0, ui2[t] + 5*Derivative[1][
      uv2][t] == 0},
3 {ui1[t], ui2[t], ui3[t], ui4[t], ui5[t], ui6[t], ui7[t], uids[t
      ], uv11[t], uv12[t], uv2[t], uv3[t], uv4[t], uv5[t], uv6[t
      ], uv7[t], uvds[t], uvgs[t], uvo[t]},
4 t]
5 ]
```

図 4.2　Mathematica における実行文

Timing 関数は引数に与えた文（図 4.2 においては DSolve 関数）を評価するのにかかる時間を計測する．実行の結果，1130.15482[秒] かかり，26602[char] の解が得られた.

4.2.3　Maple の DAE 解析能力

Maple における実行文を図 4.3 に示す.

Maple は標準で文を評価するのにかかった時間を出力する．ただし，Mathematica と Maple では数式の形式が異なるため図 4.3 にの実行結果を Mathematica 形式に直し，さらに prev[pa, 0] を pa とした文字数を結果とする．実行の結果，0.66[秒] かかり，

```
1 lprint(
2 dsolve(
3 {30*ui3(t) = uv3(t), 3*ui4(t) = uv4(t), ui5(t) = 10*diff(uv5(t)
      , t), ui4(t) + ui5(t) = uids(t), 30*ui6(t) = uv6(t), ui1(t)
       + ui6(t) = ui3(t), 10*ui7(t) = uv7(t), ui2(t) + ui7(t) =
      uids(t), uids(t) = 10*uvgs(t), uv11(t) = -10*ui1(t), uv3(t)
       = uv4(t) + uvgs(t), uv4(t) = uv5(t), uv3(t) + uv6(t) = 12,
       sin(t*pa) + uv11(t) + uv12(t) = uv3(t), uvo(t) = -30*ui2(t
      ), uv2(t) + uvo(t) = uv4(t) + uvds(t), uv2(t) + uv7(t) +
      uvo(t) = 12, ui1(t) + 5*diff(uv12(t), t) = 0, ui2(t) + 5*
      diff(uv2(t), t) = 0}
4 )
5 );
```

図 4.3　Maple における実行文

9347[char] の解が得られた.

4.2.4　結果

Maple は Mathematica よりも短時間で結果を得ることができた．さらにに Maple は Mathematica よりも短い数式として解を得ることができているとわかる．以上のことから Maple は DAE を解く速度および数式の簡約化において Mathematica よりも優れており HyLaGI に導入する意義があると言える.

4.3　Maple を HyLaGI に導入する（手法 3）

Maple は OpenMaple という C 言語用の API を提供しており, C++ で開発されている HyLaGI から利用することができる．本研究では Mathematica の組み込み関数 DSolve で行う処理のみを Maple の組み込み関数 dsolve に置き換えた HyLaGI を用いて実験を行う．Maple の組み込み関数 dsolve は DAE を解くことができるが Mathematica とは異なり，（簡単な微分方程式を除き）一般解しか求めることができない．そこで手法 2 における Mathematica の DSolve で一般解を求める部分を Maple での dsolve で解く（手

法 3 とする）ことで高速化を目指す．作業内容としては API を利用するためのヘッダのインクルードおよび環境整備，数式を Mathematica 形式と Maple 形式に相互変換する処理，Mathematica が解こうとしている DAE を Maple に入力する処理，Maple の結果を Mathematica 入力する処理が挙げられる．実装内容の詳細は早稲田大学上田研究室が GitHub 上で公開している HyLaGI の maple ブランチのコミット履歴にて確認できる．（https://github.com/HydLa/HyLaGI/commits/maple）

第 5 章

改善された HyLaGI の性能評価

この章では 3 章および 4 章で説明した改善を行った HyLaGI を用いて共通の HydLa プログラムを実行し，実行時間を計測した．また，実行結果を用いて HyLaGI によるパラメタを含んだ制度保証計算ならではの解析の例を示す．

5.1 実行環境

CPU:AMD Ryzen Threadripper 1950X 3.4GHz
メモリ:64GB
OS:Ubuntu 18.04 LTS
Mathematica のバージョン:11.3
Maple のバージョン:Maple2015
（4.2.1 節と同じである．）

5.2 解析する電気回路例題

4.2.2 節でも用いた MOSFET 増幅回路とトランジスタ増幅回路におけるインパルス応答 [7] と角周波数特性を調べる例題で性能の比較を行う．

5.2.1 解析する電気回路

図 5.1 と 5.2 は早稲田大学 2015 年度の講義「電子回路」のレポート課題から引用した．図 5.1 においては $r_g = 10[\mathrm{k}\Omega]$ とし，またゲート電圧と MOSFET を流れる電流は完全に比例するとしその倍率は $10[\mathrm{mS}]{=}10[\mathrm{mA}V^{-1}]$ とする．図 5.2 においては $r_g = 5[\mathrm{k}\Omega]$ と

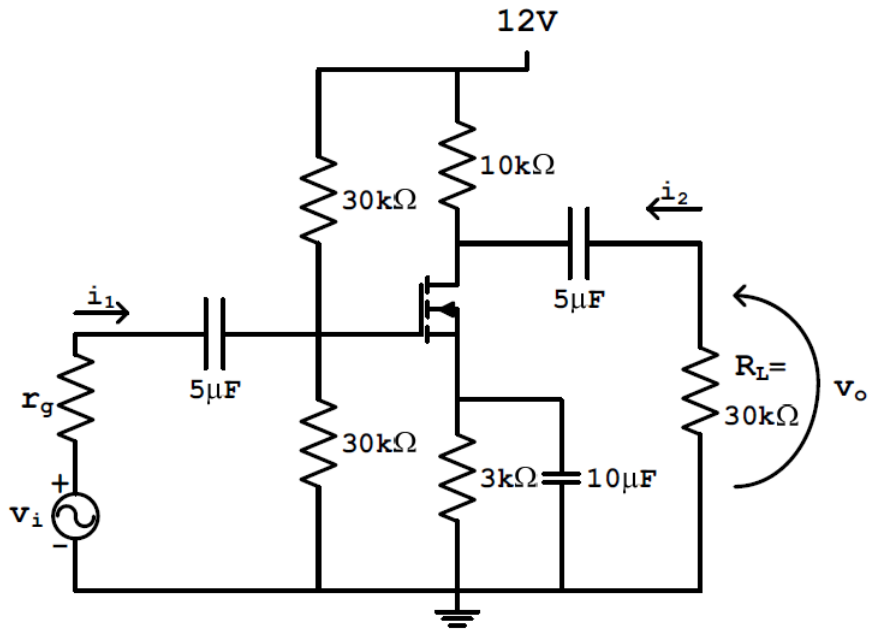

図 5.1　MOSFET を使用した信号増幅回路（再掲）

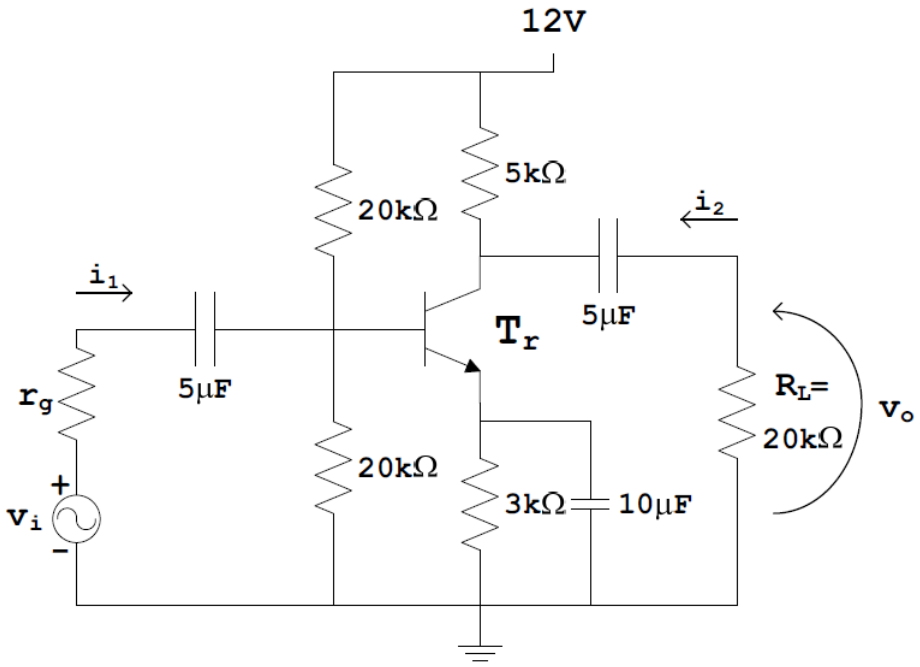

図 5.2　トランジスタを使用した信号増幅回路

し，またベース電流とコレクタ電流は完全に比例するとしその倍率は 120 とする．

5.2.2　使用する HydLa プログラム

MOSFET 増幅回路とトランジスタ増幅回路のそれぞれにインパルス応答と角周波数特性を調べる HydLa プログラムを作るので合計 4 つの HydLa プログラムができる．例として MOSFET 増幅回路のインパルス応答を調べる Hydla プログラムを図 5.3 に，トランジスタ増幅回路の角周波数特性を調べる Hydla プログラムを 5.4 に示す．

```
1  TIMER <=> timer=0 & [](timer'=1).
2  VIN1 <=> [](vin'=0).
3  VIN2 <=> [](timer=1 => vin=1/eps).
4  VIN3 <=> [](timer=1+eps => vin=0).
5  PAR(a,b,c) <=> a<b<c & [](b'=0).
6  R(v,i,r) <=> [](v=r*i).
7  L(v,i,l) <=> [](v=l*i').
8  C(v,i,c) <=> [](c*v'=i).
9  INIT <=> vin=0 & v12'=0 & v2'=0 & v5'=0.
10 CIRCUIT <=> [](vin+v11+v12=v3 & v3+v6=12 & vo+v2+v7=12 & v4=v5
       & v4+vgs=v3 & v4+vds=vo+v2 & i1+i6=i3 & ids=i2+i7 & ids=i4+
       i5).
11 MOS(v,i,g) <=> [](i=g*v).
12
13 TIMER,
14 VIN1 << (VIN2, VIN3),
15 PAR(0,eps,1),
16 R(v11,-i1,10), C(v12,-i1,5), R(v3,i3,30), R(v6,i6,30), R(vo,-i2
       ,30), C(v2,-i2,5), R(v7,i7,10), R(v4,i4,3), C(v5,i5,10),
17 INIT,
18 CIRCUIT,
19 MOS(vgs,ids,10).
```

図 5.3　MOSFET 増幅回路のインパルス応答を調べる Hydla プログラム

```
1  VIN <=> [](vin=sin(a*t)).
2  PAR(a,b,c) <=> a<b<c & [](b'=0).
3  R(v,i,r) <=> [](v=r*i).
4  L(v,i,l) <=> [](v=l*i').
5  C(v,i,c) <=> [](c*v'=i).
6  INIT <=> v12'=0 & v2'=0 & v5'=0.
7  CIRCUIT <=> [](vin+v11+v12=v3 & v3+v6=12 & vo+v2+v7=12 & v4=v5
       & v4=v3 & v4+vce=vo+v2 & i1+i6=i3+ibe & ice=i2+i7 & ibe+ice
       =i4+i5).
8  TRANSISTOR(v,i,g) <=> [](i=g*v).
9
10 VIN,
11 PAR(0.5,a,1.5),
12 R(v11,-i1,5), C(v12,-i1,5), R(v3,i3,20), R(v6,i6,20), R(vo,-i2
       ,20), C(v2,-i2,5), R(v7,i7,5), R(v4,i4,3), C(v5,i5,10),
13 INIT,
14 CIRCUIT,
15 TRANSISTOR(ibe,ice,120).
```

図 5.4　トランジスタ増幅回路の角周波数特性を調べる Hydla プログラム

補足として，図 5.3 のプログラムを HyLaGI で実行する際，epsilon mode[5] を使用することにより結果に含まれるパラメタ eps を正の向きから 0 に近づけたときの変数の軌道を調べることができる．

5.2.3　使用する HyLaGI のバージョンについて

本研究の効果を確認するために 4 つの HyLaGI のバージョンによる実行時間を比較する．3 章で述べた問題点を改善する前の HyLaGI および手法 1，手法 2，手法 3 を実装した HyLaGI を用いる．

5.2.4　実行結果

実行結果を表 5.1 に示す.

表 5.1　入力電圧の周波数と増幅率の関係

	HyLaGI1	HyLaGI2	HyLaGI3	HyLaGI4
MOSFET 増幅回路のインパルス応答	72.552474[s]	4.617226[s]	4.896299[s]	5.750176 [s]
トランジスタ増幅回路のインパルス応答	120.084725[s]	42.462462[s]	46.245005[s]	（求解不可）
MOSFET 増幅回路の角周波数特性	（求解不可）	1150.021564[s]	1155.830337[s]	10.960628[s]
トランジスタ増幅回路の角周波数特性	（求解不可）	47.995069[s]	47.744469[s]	（求解不可）

結果を見ると，問題点を改善する前の HyLaGI では角周波数特性を調べるプログラムは実行できず，実行できたインパルス応答を調べるプログラムも何らかの改善をした HyLaGI よりも実行時間が長かった．3 章で述べた手法 1 および手法 2 を実装した HyLaGI の間には大きな差が無いと言える．Maple を用いた場合は MOSFET の角周波数特性については大幅な時間短縮ができた．MOSFET の角周波数特性についてはもともと短時間で解けていたため計算時間の短縮よりも Maple との通信のオーバーヘッドが大きかったと言える．トランジスタ増幅回路は途中で Mathematica の動作が止まり，解が求められなかった．原因としてはトランジスタは MOSFET と異なり増幅元からの電流を引き込むため式が複雑化する．更に Maple を用いるにあたって従来局所変数として保存していた非常に長い変数をグローバル変数として保存せざるを得なかったので Mathematica の負荷が増大した．この 2 点が重なり，Mathematica の動作が止まってしまったと考えられる．同一の HydLa プログラムについて HyLaGI のバージョンが異なっても解が得られたものについては同一の解が得られた.

5.3　HyLaGI ならではの電気回路解析

HyLaGI が出力した解は数式となっているため式中に含まれる時刻 t やパラメタについて特定の値をが代入するだけでなく，極限を取ることで「十分な時間が経過すると」「十分大きい」という言葉で表されるような状態について定量的に説明ができる．

5.3.1　インパルス応答を調べる

図 5.1 の MOSFET 増幅回路におけるインパルス応答を調べたいとする．図 5.3 の HydLa プログラムを実行し，パルス信号が発生した後の vo の軌道を調べる．図 5.5 にパルス発生後の vo の軌道を示す．

```
1  E^(1/200+t*(-31)/30)*(93784*E^((9+3076*t+24*eps)*1/3000)+65365*
       E^(t*617/600)+1395000*E^(eps*31/30+617/600)+E^(617/600)
       *(-1395000)+E^((617*t+3*eps)*1/600)*(-65365)+E^((9+3076*t)
       *1/3000)*(-93784))*eps^(-1)*15/474473
```

図 5.5　パルス発生後の vo の軌道

図 5.5 は発生時刻が $t = 1$ から $t = 1 + eps$ の間，大きさが $\frac{1}{eps}$ のパルス信号に対する応答であるから，$eps \to +0$ の極限を取ることで発生時刻が $t = 1$ のインパルス応答（言い換えると $\delta(t-1)$) に対する応答）となる．これを図 5.6 に示す．（epsilon mode で実行すると自動で極限を求めてくれる）

$$-\frac{51e^{\frac{1-t}{200}}}{4936}+\frac{456e^{\frac{1-t}{125}}}{19225}+\frac{21622500e^{-\frac{31}{30}(-1+t)}}{474473}$$

図 5.6　発生時刻が $t = 1$ のインパルス応答

図 5.6 において t を $t+1$ に置き換えると，$t = 0$ のインパルス応答（言い換えると $\delta(t)$) に対する応答）となる．これを図 5.7 に示す．

$$\frac{21622500e^{-31t/30}}{474473} + \frac{456e^{-t/125}}{19225} - \frac{51e^{-t/200}}{4936}$$

図 5.7　発生時刻が $t = 0$ のインパルス応答

5.3.2　角周波数特性を調べる

図 5.2 のトランジスタ増幅回路における角周波数特性を調べたいとする．図 5.4 の HydLa プログラムを実行し，vo の軌道を調べる．図 5.8 に vo の軌道を示す．

```
1 (6000*a*(-46223846*(139129 + 625*a^2*(1225369 + 90000*a^2))*E
    ^(((5531 + 5*Sqrt[1216417])*t)/3000) + (1 + 15625*a^2)
    *(22500*a^2*(169209686785 + 153405767*Sqrt[1216417] +
    (169209686785 - 153405767*Sqrt[1216417])*E^((Sqrt[1216417]*
    t)/300)) + 139129*(23111923 - 11941*Sqrt[1216417] +
    (23111923 + 11941*Sqrt[1216417])*E^((Sqrt[1216417]*t)/300))
    ) - 1150122273500*a*E^(((1111 + Sqrt[1216417])*t)/600)*(30*
    a*(2107 + 3448125*a^2)*Cos[a*t] + (-373 + 125*a^2*(9979 +
    225000*a^2))*Sin[a*t])))/(2300244547*(1 + 15625*a^2)
    *(139129 + 625*a^2*(1225369 + 90000*a^2))*E^(((1111 + Sqrt
    [1216417])*t)/600))
```

図 5.8　vo の軌道

十分に時間が経過したときの様子を調べるために図 5.8 において $t \to \infty$ で収束しない項を取り出した式を図 5.9 に示す．

$$\frac{3000000a^2\left(30a\left(2107 + 3448125a^2\right)\cos(at) + \left(-373 + 125a^2\left(9979 + 225000a^2\right)\right)\sin(at)\right)}{\left(1 + 15625a^2\right)\left(139129 + 625a^2\left(1225369 + 90000a^2\right)\right)}$$

図 5.9　$t \to \infty$ で収束しない項

図 5.9 を $\alpha\sin(at + \theta)$ の形に変形する．α の値は増幅率，θ の値は位相を示す．そのと

きの α の値を図 5.10，θ の値を図 5.11 に示す．

$$\frac{3000000\left(a^2+900a^4\right)}{\sqrt{\left(1+900a^2\right)\left(1+15625a^2\right)\left(139129+625a^2\left(1225369+90000a^2\right)\right)}}$$

図 5.10　α の値

$$\arctan\left(\frac{30a\left(2107+3448125a^2\right)}{-373+125a^2\left(9979+225000a^2\right)}\right)$$

図 5.11　θ の値

さらに，極限 $a\to\infty$ を求めることで，入力信号の（角）周波数が十分に大きい場合の増幅率や位相を調べることができる．$a\to\infty$ のとき，$\alpha\to 96$ や $\theta\to 0$ である．このことから図 5.2 のトランジスタ増幅回路の増幅率は 96 であることに加えて，周波数がどれだけ大きくなっても増幅率が発散することはないという安全性を説明できる．

第 6 章

まとめと今後の課題

6.1　まとめ

HyLaGI に DAE を解くためのアルゴリズムを実装し，解けるはずの問題が解けないという問題を解決した．さらに，Maple を HyLaGI に導入することで Mathematica のみでは解析に時間のかかる例題を短時間で解くことができるようになったが，複雑すぎる数式を扱うとシステムへの負荷が大きくなり動作が停止してしまうことが明らかになった．パラメタを用いて特定の時刻やパラメタの値だけでなく，値が「十分大きい」,「十分 0 に近い」という状態を定量的に説明したり，安全性の説明に応用することができることも明らかになった．

6.2　今後の課題

Maple を用いるために数式を Mathematica におけるグローバル変数に保存するのではなく C++ 上の文字列として保存するようにすることで処理の軽減を目指すことが挙げられる．また，Maple で使用できる SyNRAC モジュールに含まれる QE ソルバを用いて不等式制約の解を求めることで，いままで HyLaGI がとくこのとできなかった HydLa プログラムの解を求めることができるようになることが期待される．しかしあらゆる方法を試して実行時間を短縮しようとしても Mathematica も Maple もブラックボックスなシステムであるため実行時間が短縮される根拠は不透明である．そのため今後 HyLaGI を信頼性のあるシステムとして開発して行くことを考えると，Mathematica や Maple の代わりに SageMath などのオープンソースな数式システムをバックエンドとして利用できるようにすることが重要といえる．

謝辞

電気回路の例題はハイブリッドシステムの研究において基本的で抑えるべき問題であることが分かる論文を提示していただき，研究の方針を指導してくださった上田和紀教授に深く感謝いたします．電気回路の例題が HydLa で書けるというアイデアは安生さんの発表からいただきました．また，微分代数方程式が線型の微分方程式が異なるということを教えてくれたのは佐藤君でした．また，Maple 形式数式をの Mathematica 形式に変換するプログラムを作ってくれた山田君のおかげで Maple の導入ができました．HydLa 班以外のメンバーからも研究に役立つ情報をたくさん頂きました．HydLa 班の先輩を始め，研究室の環境の利用法の指導や，ゼミで議論をしてくださった上田研の皆さんのおかげで研究テーマに関する話題を論文を書ける形にまとめることができました．

2020 年 1 月　渋井 隆弘

参考文献

[1] 上田和紀，石井大輔，細部博史，ハイブリッド制約言語 HydLa の宣言的意味論，コンピュータソフトウェア，Vol. 28 No. 1, 2011, pp. 306–311.

[2] 松本翔太 : Validated Simulation of Parametric Hybrid Systems Based on Constraints, 博士論文，早稲田大学大学院基幹理工学研究科，2017.

[3] 岩田覚，高松瑞代 : 電気回路の混合解析における微分代数方程式の指数最小化，数理解析研究所講究録，Vol. 1629, pp. 104–114 (2009).

[4] IEEE Access，Edward A．Lee，Constructive Models of Discrete and Continuous Physical Phenomena，Vol. 2, 2014, pp. 797–821.

[5] 若槻祐彰 : ハイブリッド制約処理系 Hyrose における微小値を用いたシミュレーション手法，卒業論文，早稲田大学基幹理工学部情報理工学科，2014.

[6] オーム社, 渋谷道雄, 回路シミュレータ LTspice で学ぶ電子回路, 第 2 版, 2016, p. 82.

[7] 朝倉書店，永野宏治，信号処理とフーリエ変換，初版, 2014, pp. 95–101.

[8] Analog Devices: LTspice, (online), https://www.analog.com/jp/design-center/design-tools-and-calculators/ltspice-simulator.html, 2019-01-10.

[9] Wolfram Mathematica，https://www.wolfram.com/mathematica/index.ja.html，2017.

[10] サイバネット: 数式処理ベース STEM コンピューティング・プラットフォーム Maple, (online), http://www.cybernet.co.jp/maple/, 2019-01-10.

発表論文

[1] 渋井隆弘，上田和紀，ハイブリッドシステムモデリング言語 HydLa を用いた DAE システムの解析，情報処理学会第 81 回全国大会，2019.